Which OS for IEC 62304 Medical Systems? A Comparison of Required Development Effort with Linux and the QNX Neutrino RTOS

Chris Hobbs, Senior Developer, Safe Systems
chobbs@qnx.com

Abstract

This paper is intended for anyone who needs to select an OS for a safety-critical medical system. It attempts to give information that will help with estimations of the real cost of choosing a Linux or QNX OS. It lists requirements identified by standards such as IEC 62304, ISO 14971 and IEC 61508, and provides comparative estimates of the effort required to meet these requirements. These estimates are for initial certification and pre-approval, subsequent re-certifications following OS upgrades, and ongoing costs.

Introduction

A recurring question in discussions around software for medical systems is "Why not Linux?" Linux has a long history of successful implementations in many industries and a large developer community from which talent can be recruited. Moreover, it is available in many different flavors for innumerable platforms and it is freely available. There is no OS vendor to pay.

As an OS programmer and user, I admire Linux and know that for many projects it can be an excellent choice. Indeed, I use nothing else on my laptop and desktop computers, whether at work or at home. As a programmer working for an OS vendor to medical device manufacturers (amongst many others), however, I have wondered whether a "free" OS is as good a choice for a safe medical system as is our OS. As a specialist in safe software system design and validation, I am a natural skeptic, and require evidence for any claim I am expected to accept.

Responsibility for Safe Operation

Note that, in accordance IEC 62304, paragraph B.1.2, whether the OS be Linux, the QNX Neutrino RTOS or some other operating system, its safe operation is the responsibility of the developer of the medical device.

As a skeptic, and assuming that QNX Software Systems‟ customers should be as skeptical as I am, I decided to present here informal[1] evidence to

[1] "Informal" because the evidence is a) not a formal proof, and b) estimates based on my knowledge of safe software design and not on a large-scale comparative cost study. Such a study would be rather difficult to carry out, even if medical device manufacturers were willing to share information about their development costs: too

support my estimates of the additional costs of building a safe software system with Linux rather than with the QNX® Neutrino® RTOS.

Criteria for comparison

At the highest level, the arguments for or against using Linux rather than a proprietary OS can be grouped into two categories: OS characteristics and cost.

OS characteristics

OS characteristics include performance and dependability (availability and reliability). This is not the subject of this paper. For more information see:

- Chris Hobbs, "Clear SOUP and COTS Software for Medical Device Development" (2011), for a discussion of the implications for safety certification and pre-market approval of using software of uncertain provenance (SOUP) and commercial-off-the-shelf software (COTS) <http://www.qnx.com/download/feature.html?programid=22793>
- Justin Moon, "Choosing an RTOS for Remote Care Medical Devices" (2011) for a discussion of OS characteristics, such as dependability, that are important for safety-critical medical systems <http://www.qnx.com/download/feature.html?programid=22012>

Useful performance comparisons of real-time OSs (RTOSs) for embedded systems are available from various independent evaluators, including Dedicated Systems. Their reports are available at: <http://es2.dedicated-systems.info/>

Cost

The costs associated with developing safe software systems include all the usual suspects of software projects: software licenses, tools, staffing, etc., plus the cost of:

- building a software system that dependable enough ("sufficiently dependable") to meet its safety requirements
- building the safety case: the proofs that the software meets its dependability requirements when it is used in accordance with the conditions and limits specified in its accompanying safety manual
- getting the completed device through the pre-market approval process with the regulatory agencies in the jurisdictions where the device will be sold

This paper lists requirements identified by standards such as IEC 62304, ISO 14971 and IEC 61508, and provides comparative estimates of the effort required to meet these requirements.

many other variables, such as system complexity, developer skill, and management, would render the data largely meaningless.

Method and assumptions

I arrived at the estimates presented here in the same manner as are most estimates of effort for software development projects—when these are correctly done:

1. Identify requirements. Since the comparison is for the safety-related aspects of the system, only these requirements are considered.
2. Estimate the work required to meet each requirement, based on previous experience with similar systems.

I have based my analysis and estimates on the actual wording of IEC 62304 (and, where applicable, ISO 14971 and ANSI/AAMI/IEC TIR80002-1:2009) rather than my own opinion. ISO 14971 is explicitly referenced in IEC 62304.

I have concentrated on the OS itself and not the tool-chains to be used (for instance, it is difficult to demonstrate the integrity of a C compiler; it might be much harder to demonstrate the correctness of a Python interpreter).

In some cases, I have had to assume that the required work would be possible with Linux, because it is not clear that all the required documentation will actually be available. If no documentation were available, then it would simply be impractical to use Linux.

Programmer skill and experience

The time a programmer requires to design and development varies greatly depending on the programmer‟s skill and experience. Programmers‟ estimates of the time they require similarly depend on skill and experience. As every project and development manager knows—or should know—these estimates also very much depend on individual personalities.

The estimates I present here are based on how long I consider it would take me to complete the work to meet each requirement. Note, however, that I am an OS programmer deeply familiar with Linux, and with more than 30 years of experience, a good part of which has been focused on developing safe systems. A less experienced programmer would likely require more time.

Finally, in the past I have been told by my managers that, like so many programmers, I tend to under-estimate required development effort. I have therefore added a conservative 30 percent to my estimates.

Estimation of effort

Table 1 summarizes the activities that would be required for the initial deployment of a Linux-based system in a device requiring safety certification or pre-market approval.

Activity	Reference	Effort (engineer months)
Produce risk analysis	3, 7 & 9	18
Produce functional and performance requirements	4	4

Validation specification for functional and performance requirements	4	3
Validation of functional and performance requirements	4	4
Identify unnecessary components	8	4
Remove unnecessary components	8	12
Identify safety aspects of Linux	10	4
Total		**49**

Table 1. Tentative estimate of the additional effort required for the initial deployment of a Linux-based safe system. The numbers in the Reference column refer to the item numbers in the "References and activities" section below. Refer to this section for more detail about each activity, including references to the relevant standards.

For simplicity, the list includes only activities that would be necessary with the Linux system but which would not be required with a system using the QNX Neutrino RTOS Safe Kernel. Activities for which the effort with Linux OS and the QNX OS are approximately equivalent are not included.

In addition to this initial deployment effort, continuous background activities will be needed. These activities are listed with estimates in Table 2.

Activity	**Reference**	**Effort (full-time heads per month)**
Monitor Linux changes and decide when to deploy them	2	0.25
Monitor Linux bug reports and perform impact analyses	2 & 5	1
Total		**1.25**

Table 2. Tentative estimate of the additional effort required for the ongoing support of a Linux-based safe system.

References and activities

This section presents items in the safety standards that are relevant to the estimates, along with brief explanations of their implications for the estimates.

1. Both Linux and the QNX Neutrino RTOS are SOUP

IEC 62304, paragraph **3.29** defines SOUP[2] as a "[1] software item that is already developed and generally available and that has not been developed for the purpose of being incorporated into the medical device (also known as „off-the-shelf software") or [2] software previously developed for which adequate records of the development processes are not available."

[2] "Software Of Unknown Provenance", or "Software Of Unknown Pedigree"

Linux is SOUP on both of these counts; the QNX Neutrino RTOS Safe Kernel is SOUP on the first count only.[3]

2. Change management

IEC 62304, paragraph **5.1.1** states that the software development plan must address “software configuration and change management, including SOUP configuration items and software used to support development”.

This requirement is equally applicable to Linux and the QNX Neutrino RTOS. Producing a change management plan for Linux would be somewhat more difficult than for the QNX Neutrino RTOS, however: when would new releases be incorporated into the product, and what level of validation would be required on them before incorporation?

3. Software risk management planning

IEC 62304, paragraph **5.1.7** stipulates “the manufacturer shall include or reference in the software development plan, a plan to conduct the activities and tasks of the software risk management process, including the management of risks relating to SOUP.”

This requirement applies to both Linux and the QNX Neutrino RTOS, however:

- QNX Software Systems can provide the developers working with its OS a risk analysis in accordance with ISO 14971 (incorporating the information from TIR80002), and the QNX Neutrino RTOS Safe Kernel IEC 61508 certification brings with it an upper bound on failure rate.
- With Linux, risk analysis would need to be produced by the development organization, and it is not clear how it could be produced without a failure rate figure being available.

For the purpose of the comparative estimate I have assumed that software risk management planning is possible with Linux. It is not clear exactly how this could be done.

[3] At QNX Software Systems, we call SOUP on only the first count “clear SOUP” because we make our processes and records available for scrutiny by customers.

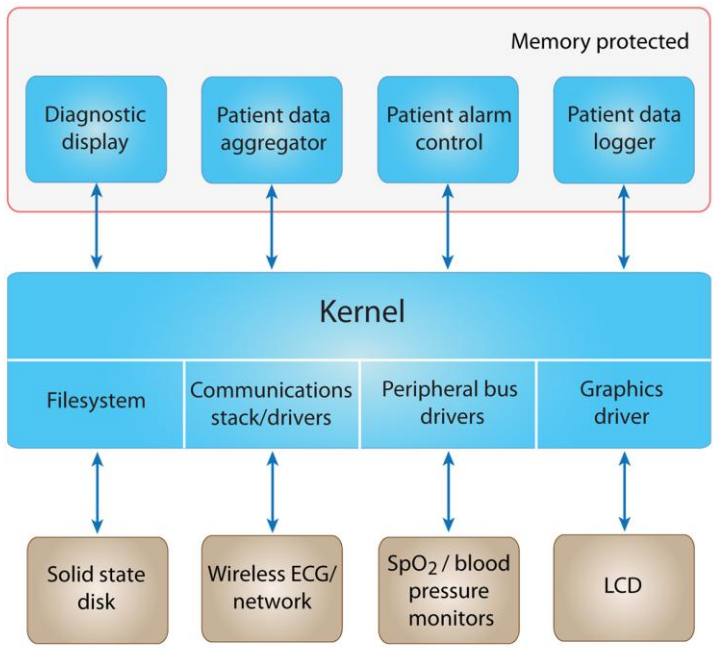

Figure 1.A diagram showing a hypothetical medical monitoring device. The OS has a monolithic architecture, such as that used by Linux. Drivers, file systems, the communications stack, etc. must all be included in the risk analysis because they can corrupt the kernel.

However, I have assumed that the bug reports for the particular Linux distribution could be read and categorized, and the level of fault reporting estimated. Additionally, I have assumed that the level of stability of the software could be measured (number of modules changed, etc.), and that from this and an approximation of the usage of the Linux distribution, it would be possible to estimate a failure rate per hour of operation.

Further, the monolithic architecture of the Linux OS would make this work much more difficult and time-consuming than the corresponding work for the QNX Neutrino RTOS Safe Kernel, which uses a microkernel architecture. With Linux, all fault reports in the file system, communications stacks, drivers, etc. would also have to be analyzed—a rather arduous task. With QNX this analysis is unnecessary because these components can be easily restarted and cannot affect the system.

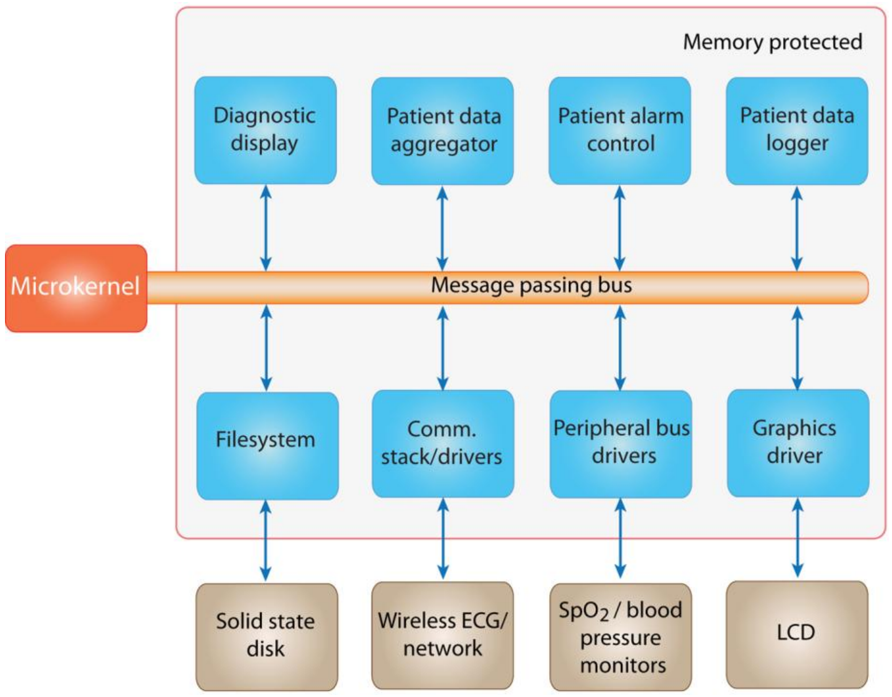

Figure 2. The device shown in Figure 1. With microkernel OS architecture, drivers, filesystems, the communications stack, etc. are outside the kernel. They cannot corrupt the kernel and may be excluded from the risk analysis of the kernel.

4. Functional and Performance Requirements

IEC 62304, paragraph **5.3.3** states: "If a software item is identified as SOUP, the manufacturer shall specify functional and performance requirements for the SOUP item that are necessary for its intended use."

The problem here is specifying the functional requirements. QNX specifies the safety claims it makes for the QNX Neutrino RTOS Safe Kernel in its functional safety requirements; and this OS‟s certification to IEC 61508 indicates that it meets them. These claims can be brought directly forward into the device manufacturer‟s requirements.

For Linux, the functional safety requirements would have to be defined. Someone would then have to carry out the necessary validation to ensure that the Linux OS complies with these requirements.

5. Published SOUP anomaly lists

IEC 62304, paragraph **6.1** states that: "if failure or unexpected results from SOUP is a potential cause of the software item contributing to a hazardous situation, the manufacturer shall evaluate […] any anomaly list published by the supplier of the SOUP item relevant to the version of the SOUP item used in the medical device to determine if any of the known anomalies result in a sequence of events that could result in a hazardous situation."

This requirement is augmented in **TIR80002**, paragraph **3.4.1**: "If software of unknown provenance (SOUP) is used, then actively monitoring and evaluating publicly available anomaly lists and information about the SOUP field performance should be planned. *Where possible, this should be supported by a contractual agreement with the SOUP supplier at the time of SOUP acquisition.*" (my emphasis)

A failure of the operating system, whether Linux or the QNX Neutrino RTOS, could in some cases result in a hazardous situation in almost all devices. However, in the case of Linux, an examination of the anomaly list would require a manual inspection of the open problem reports relating to Linux. The number of open Linux problem reports oscillates between 5000 and 8000, so inspecting them is not a trivial activity. Further, this activity must be continued after product release for as long as the device is operational. In contrast, under the QNX problem report review process (QMS 0531) and impact analysis process (QMS 0532), QNX Software Systems already performs this analysis for both the QNX Neutrino RTOS Safe Kernel and the QNX Neutrino RTOS for medical devices.

To follow the advice of TIR80002, the device manufacturer should have a contractual agreement with the SOUP supplier. This is of course possible for a Linux system as there are many vendors selling and supporting their particular Linux variants, but it is easier to do for the QNX Neutrino RTOS Safe Kernel, which a vendor has supported from its inception.

6. Responsibility for SOUP

IEC 62304, paragraph **B.1.2** states that: "when the medical device system architecture includes an acquired component […] the acquired component becomes the responsibility of the manufacturer and must be included in the risk management of the medical device."

Thus, if Linux is used for a medical device, the device manufacturer must assume responsibility for the full risk management of the Linux variant used. It is unclear, however, how the manufacturer could take on such a responsibility, having neither control over the development process nor the use of a certified product.

7. Failure modes of SOUP

TIR80002, paragraph **6.2.3** states: "The higher the potential risks of the medical device, the more closely potential failure modes of SOUP should be analyzed and risk control measures identified. […] Nor is there adequate internal design information available to identify all the potential hazards caused by SOUP."

To meet this recommendation when using Linux, a failure mode analysis of Linux is required. This is an enormous undertaking because, the Linux kernel includes drivers, file systems, the communications stack, etc., and a failure in any of these components can cause a kernel crash. See Figure 1.

A failure mode analysis for the QNX Neutrino RTOS Safe Kernel already exists. In addition, with the QNX microkernel architecture, drivers, file systems, the communications stack, etc. all reside outside the kernel, a design that protects the kernel from crashes in these components. Compare Figure 2 with Figure 1.

8. Removing unnecessary SOUP components

TIR80002, section **9** notes that “Released SOUP patches or updates […] may include additional functionality that is not essential for the safety and effectiveness of the medical device. These SOUP updates should be analyzed for excessive components which can be eliminated from the medical software release to avoid unexpected changes which may lead to a hazardous situation.” This point is stressed again in the same standard, table **B.2**: “Use only the SOUP features required: remove all others”.

Linux is a *very* large system and includes many drivers and other components that may be unnecessary for the continued safe operation of the medical device. Where possible, in accordance with this paragraph, these drivers and components will need to be identified and removed. Equally daunting, removing all the non-essential features of Linux would lead to an endless cycle of “clone-and-own” work. Every Linux upgrade would require removal of unused features before it could be used in the medical device.

Since the QNX Neutrino RTOS Safe Kernel is a microkernel, it does not carry any of this extra baggage. No drivers or components need to be removed, reducing overall project work for the device manufacturer.

Implications for failure analysis

The work required to remove all unneeded components from Linux in order to use it in a safe medical device may in fact prove relatively minor compared to the effect of these removals on safety certifications and pre-market approvals.

Removing parts of the Linux kernel would invalidate much of the failure analysis created to meet the requirements noted under point 7. To avoid invalidating the failure analysis, the device manufacturer would need to carry out the work of removing the unneeded components and performing the analysis in parallel. Both would require the attention of highly skilled staff familiar with the Linux kernel and support packages, and would have to be repeated for each upgrade.

9. Failure modes and frequency of SOUP

ISO 14971, paragraph **4.3** (called up in **TIR80002**, table **B**) expressly warns the designer against omitting the failure of SOUP in the fault tree (or FMECA) for the medical device. Again, this warning stresses the need for a fault tree.

It is unclear where the information on the failure rate of Linux would come from. Linux obviously has an enormous amount of confidence-in-use data but, to my knowledge, neither the number of hours nor, more importantly, the failure modes and numbers have been gathered.

Validity of confidence-in-use statistics

Confidence-in-use statistics must refer to the precise build of the software that the device manufacturer is using. If the unused features of Linux are removed (see point 8) then that makes a unique version of Linux with no associated confidence-in-use figures.

10. Identifying safety-related aspects of SOUP architecture

TIR80002, table **C.1** identifies, in the context of SOUP, the pitfall of: “Failure to identify safety-related aspects of the architecture, resulting in unknown safety risks when these architectural elements are subsequently changed or eliminated.”

The safety-related aspects of the QNX Neutrino RTOS Safe Kernel architecture have been identified as part of the Safe Kernel‟s IEC 61508 certification process. When using Linux this non-trivial work would have to be carried out by the device manufacturer.

Summary of estimated effort

In summary, I would estimate that the additional work required to deploy Linux rather than the QNX Neutrino RTOS Safe Kernel would be:

- For the initial certification and pre-approval: 49 engineer months.
- Subsequent re-certification following an upgrade to the operating system: approximately half this effort, or 24 engineer months, *if* the proper records were kept up-to-date during deployment.
- Ongoing cost (including during the first certification): 1.25 dedicated engineers for the entire life cycle of the system.

The details of how these numbers were derived are given in the section “Estimation of effort” above.

It is important to remember that, as noted above, the skills required for the estimated work are not those of a normal applications programmer; these are the skills of an operating system programmer already deeply familiar with Linux. Such programmers are rare, and they are not cheap!